JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

EXTENDED XML TREE PATTERN MATCHING: THEORY AND ALGORITHM

Mulajkar Sneha Digambarrao
P G Scholar, Department of Computer Science and Engineering, Matoshri College of Engineering Nanded.

Abstract- The New era practitioners are highly depending on the flexible open standard data structures to store and transmit the data in the B2B process, as a part of that, eXtended Markup Language(XML) Patterns are needed for efficient processing of user queries on different XML-enabled (MS-SQL Server, Oracle) Databases. The XML Document can be converted into XML Tree by using different tools like XML DOM. There are four central problems in data management: capture, storage, retrieval, and exchange of data. XML is a tool used for data exchange. Data exchange has been a long issue in information technology, but the Internet has elevated its importance. Electronic Data Interchange (EDI), the traditional data exchange standard for large organizations, is giving way to XML, which is likely to become the data exchange standard for all organizations, irrespective of size. As business and enterprises generate and exchange XML data more often, there is an increasing need for efficient processing of queries on XML data. Searching for the occurrences of a tree pattern query in an XML database is a core operation in XML query processing. Prior works demonstrate that holistic twig pattern matching algorithm is an efficient technique to answer an XML tree pattern with parent-child (P-C) and ancestor-descendant (A-D) relationships, as it can effectively control the size of intermediate results during query processing. However, XML query languages (e.g. XPath, XQuery) define more axes and functions such as negation function, order-based axis and wildcards.Here we research a large set of XML tree pattern, called extended XML tree pattern, which may include P-C, A-D relationships, negation functions, wildcards and order restriction. We establish a theoretical framework about "matching cross" which demonstrates the intrinsic reason in the proof of optimality on holistic algorithms. Based on our theorems, we propose a set of novel algorithms to efficiently process three categories of extended XML tree patterns. A set of experimental results on both real-life and synthetic data sets demonstrate the effectiveness and efficiency of our proposed theories and algorithms.

Index Terms—Query processing, XML/XSL/RDF, algorithms, tree pattern.

1. INTRODUCTION

As the Trend change, the conventional business is transmitting into the form of e-business through the Internet, which is known as B2B (Business-to-Business). It is very clear for us the B2B problem are all website based, same of the B2B websites are company websites, product supply and procurement exchanges, specialized or vertical industry portal, Information sites etc. For any type of e-business the websites is essential in this context the web information is presented in xml format and XML Document contain all the XML components of websites and the determination of XML Patterns are needed to solve the problems in B2B sites and to get more performance. In this paper we present the core concepts of Query languages. XML Trees can be Ordered and Unordered Trees. The present XML Trees can understand with a good labeling schemes[5].XPath[1], XQuery[2] are different query languages for XML. We also present the experimental results for pattern search with using the XMLBuilder and SAX parser in Java. Efficient matching of XML tree patterns has been widely considered as a core operation in XML query processing. In recent years, many methods [3-4, 9, 11, 13, 25], have been proposed to match XML tree queries efficiently. In particular, Khalifa et al. [1], proposed a stack-based algorithm to match binary structural relationship including Parent- Child (P-C) and Ancestor-Descendant (A-D) relationships. The limitation of their method is that the size of useless intermediate results may become very large, even if the final results are small. Bruno et al. [3], proposed a novel holistic twig join algorithm named Twig Stack, which processes the tree pattern holistically without decomposing it into several small binary relationships. Twig Stack guarantees that there Ancestor- Descendant (A-D) relationships. In other words, Twig Stack is optimal for tree pattern queries with only A-D edges [8]. Many other recent works then examine how to enlarge the optimal query class of holistic algorithms [14], to speed up performance using indexes [5, 11], to devise new data streaming strategies [6], and to propose efficient and dynamic labeling schemes [16].

1.1 Background and Motivations

Previous algorithms focus on XML tree pattern queries with only P-C and A-D relation- ships. Little work has been done on XML tree queries which may contain wildcards, negation function, and order restriction, all of which are frequently used in XML query languages such as XPath and XQuery. In this paper, we call an XML tree pattern with negation function, wildcards, and/ or order restriction as extended XML tree pattern. Fig. 2, for example, shows four extended XML tree patterns. Query (a) includes a wildcard node "*", which can match any single node in an XML database. Query (b), includes a negative edge, denoted by ":". This query finds A that has a child B, but has no child C. In XPath language [2], the semantic of negative edge can be presented with "not" boolean function. Query (c), has the order restriction, which is equivalent to an XPath "//A/B[followingsibling:: C]." The "< " in a box shows that all children under A are ordered. The semantics of order-base tree pattern is captured by a mapping from the pattern nodes to nodes in an XML database such that the structural and ordered relationships are satisfied. Finally, Query (d) is more complicated, which contains wildcards, negation function, and order restriction.

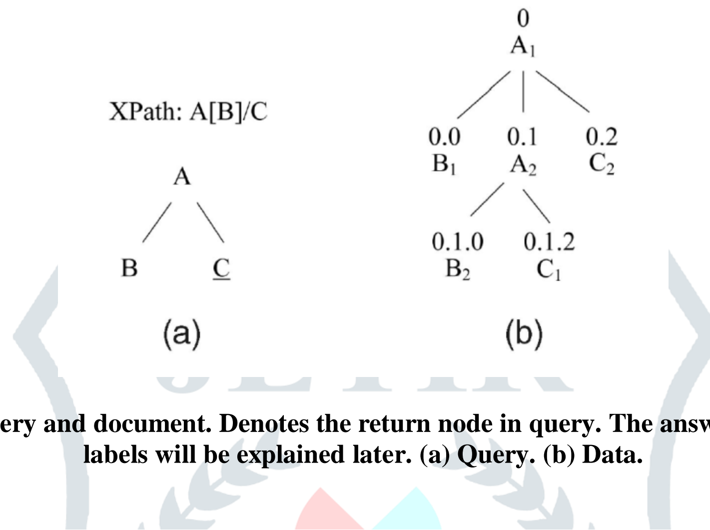

Fig. 1. Example XML tree query and document. Denotes the return node in query. The answers are C1 and C2. The digital labels will be explained later. (a) Query. (b) Data.

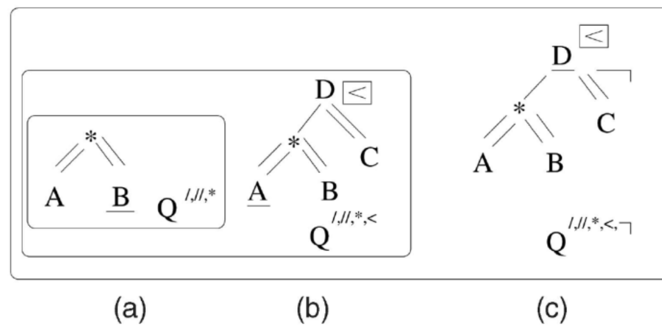

Fig. 2. Three categories of extended XML tree patterns and example optimal queries.

Therefore, TwigStack is optimal for queries with only A-D edges. Another algorithm TwigStackList [14] enlarges the optimal query class of TwigStack by including PC relationships in nonbranching edges. A natural question is whether the optimal query class of TwigStackList can be further improved. Hence, the current open problems include

1) How to identify a larger query class which can be processed optimally

2) How to efficiently answer a query which cannot be guaranteed to process optimally.

Note that earlier works in [8], [21] already showed that no algorithm is optimal for queries with any arbitrary combinations of A-D and P-C relationships. This paper explores the challenges and shows the promise of a novel theoretical framework called “matching cross” to identify a large optimal query class for posing extended XML tree queries. Return nodes in twig pattern queries.

1.2 Motivations

The New era practitioners are highly depending on the flexible open standard data structures to store and transmit the data in the B2B process, as a part of that, eXtended Markup Language(XML) Patterns are needed for efficient processing of user queries non different XML-enabled (MS-SQL Server, Oracle) Databases. The XML Document can be converted into XML Tree by using different tools like XML DOM. There are four central problems in data management: capture, storage, retrieval, and exchange of data. XML is a tool used for data exchange. Data exchange has been a long issue in information technology, but the Internet has elevated its importance. Electronic Data Interchange (EDI), the traditional data exchange standard for large organizations, is giving way to XML, which is likely to become the data exchange standard for all organizations, irrespective of size. Most of the business and enterprises generate and exchange XML data more often, so there is need for efficient processing of queries on XML data. An XML query pattern can be represented as rooted labeled tree called as twig pattern. The core operation in XML query processing is efficient matching of XML tree

patterns. The main reason why this one is introduced here is it process large XML queries. The existing algorithms process only small queries it includes only P-C and A-D relations. It process large queries with P-C, A-D, wildcards, negation function and order restriction called extended XML tree pattern. In a practical application, only part of query nodes belong to return nodes (or called output nodes interchangeably). Take the XPath "//A[B]//C" as an example, only C element and its sub tree are answers. The current "modus operandi" (e.g., [12], [3], [16]) is that they first find the query answer with the combinations of all query nodes, and then do an appropriate projection on those return nodes. Such a post processing approach has an obvious disadvantage: it outputs many matching elements of no return nodes that are unnecessary for the final results. In this paper, we develop a new encoding method to record the mapping relationships and avoid outputting no return nodes.

1.3 Objective

The occurrence of a tree pattern query in an XML database is a core operation in XML query processing. Previous algorithms search only the small tree patterns in the XML databases. Here the idea for this one is to search large tree patterns and to get optimal answers. It uses matching cross that's a theoretical framework that demonstrates the intrinsic reason in the proof of the optimality of holistic algorithms. The reason behind why we introduce matching cross is the previous algorithm's performs their work in two phases means matching and merging, at matching phase there is a chance to occur many useless intermediate results (path solutions).

1.4 Existing System

Previous algorithms focus on XML tree pattern queries with only P-C and A-D relationships. Little work has been done on XML tree queries which may contain wildcards, negation function and order restriction, all of which are frequently used in XML query languages such as XPath and XQuery. In this article, we call an XML tree pattern with negation function, wildcards and/or order restriction as extended XML tree pattern. Previous XML tree pattern matching algorithms do not fully exploit the "optimality" of holistic algorithms.

1.4.1 Disadvantages of Existing System:

1. Low search efficiency
2. Data security weak

2. REVIEW OF LITERATURE

In this paper, An XML database D is usually modeled as a rooted, node-labeled tree (in this paper, we use D to represent the database and the related tree model exchangeable without specific declaration), element tags and attributes are mapped to nodes in the trees and the edges are used to represent the direct nesting relationships. Our primary focus is on element nodes; and it is not difficult to extend our methods to process the other types of nodes, including attribute and character data. For convenience, we distinguish between query nodes and database nodes by using the term "node" to refer to a query node and the term "element" to refer to a data element in D [1].

In this paper, we use the extended Dewey labeling scheme, proposed , to assign each node in XML documents a sequence of integers to capture the structure information of documents. An extended tree query Q describes a complex traversal of the XML document and retrieves relevant tree-structured portions of it. The nodes in Q include element tags, attributes, and character data. We use "*" to denote the wildcard, which can match any single tree element. This is a simplification of Twig2Stack using simple lists and intervals given by pointers, which improves performance in practice. In the same stack structure like Twig2Stack to preserve the holisticity of the twig matches, without generating intermediate solutions. However, a considerable amount of time is taken to maintain the stack structure. [2].

In this paper, There are four kinds of query edges, which are the four combinations between (positive and negative) and (parent-child and ancestor-descen- dant). For example, in fig. 2(b), (A; B) is a positiveparent-child edge and (A; C) is a negative parent-child edge. We use a symbol ":" to denote a negative edge. There are two kinds of query node: ordered and unordered node. We use "< " in a box to denote the ordered node, otherwise it is an unordered node. The answers of a query can be represented as a set of database elements, where each element identifies a distinct match of the selected return nodes on D. For example, Fig. 5 shows an example mapping relationship between an extended XML tree pattern and a document tree and extended Dewey scheme [3].

In this paper, Extended Dewey labeling scheme is a variant scheme of the prefix labeling scheme. In the prefix labeling scheme, the root is labeled by an empty string and for a nonroot element u, labelðuÞ ¼ labelðvÞ:n, where u is the nth child of v. In Extended Dewey labeling scheme, each label provides complete information about ancestors' names and labels. For example, given an element e with label "1.2.3," prefix labeling schemes can tell us parentðeÞ¼"1:2" and grandparentðeÞ¼"1", but extended Dewey labeling scheme can also tell us the tag name of elements, say, tagðeÞ¼"A", tagðparentðeÞÞ¼"B", and tagðgrandparentðeÞÞ¼"C". In or- der to achieve this goal, paper [16] uses module function to encode the element tag information to prefix labels, and use Finite State Transducer (FST) to decode the the types information for a single extended Dewey label [4].

In this paper, But for the purpose of understanding this paper here, readers only need to know that in the extended Dewey labeling scheme, from the label of a single element, we can derive all the elements names along the path from the root to the element. And the complete path information in extended Dewey labels enables holistic algorithms to scan only leaf query nodes to answer an XML query. We adopt a structure, named label list, associated with each query node. The label list is a posting list (or inverted list) containing the extended Dewey labels of XML elements which have the same name, and all elements are ordered according to the document order. We use TA to denote the label list for query node A. There is a cursor for each list. It moves in the single direction to scan all elements once in increasing order. Each label in a list can be read only once [5].

In this paper, For a large class of queries, the main memory requirement of our algorithm is linear to the number of nodes in the longest path of XML database, which is usually small. Therefore, our solution would be scalable to a very large document with a small main memory requirement. Recall that the existing algorithms, such as TwigStack, It also have the first property. That is, they keep the single-direction scan of the document. But for the second property, those algorithms guarantee the bounded main memory for a small class of queries [6].

This paper makes the contribution to propose algorithms to achieve this property for a The idea of holistic twig join has been adopted in several works in order to make the structural join algorithm very efficient. This section is devoted to a structured review of these advances. As mentioned before, when all edges in query patterns are ancestor–descendant ones, TwigStack ensures that each roottoleaf intermediate solution contribute to the final results. However, this algorithm still cannot control a large number of intermediate results for parent–child edge query. A first improvement that can be found in the literature concerns the efficient handling of twig queries with parent–child relationships. Among the proposed approaches, TwigStack by proposing TwigStackList algorithm. This algorithm has the same performance than TwigStack for query patterns with only ancestor–descendant edges, but also produces much less useless intermediate solutions than TwigStack for queries with parent–child relationships [7].

The main technique of TwigStackList algorithm is to read more elements in the input streams and cache some of them (only those that might contribute to final answers) into lists in the main memory, so that we can make a more accurate decision to determine whether an element can contribute to the final solution or not. the suggested another algorithm, called iTwigJoin, which can be used on various data streaming strategies (e.g. Tag+Level streaming and Prefix Path Streaming). Tag+Level streaming can be optimal for both ancestor–descendant and parent–child only twig patterns whereas Prefix Path streaming could be optimal for ancestor–descendant only, parent–child only and one branch node only twig patterns assuming there was no repetitive tag in the twig patterns [8].

In this paper, an improvement strategy consists in avoiding these unnecessary computations. TSGeneric+ makes improvements on TwigStack by using XRTree to effectively skip some useless elements that do not contribute to the final results. The motivation to use XRtree is that, the ancestors (or descendants) of any XML element indexed by an XRtree can be derived with optimal worst case I/O cost. However, TSGeneric+ may still output many useless intermediate path solutions like TwigStack for queries with parent–child relationship. the TJEssential algorithm based on three optimization rules to avoid some unnecessary computations. They presented two algorithms incorporated with these optimization rules to effectively answer twig patterns in leaftoroot combining with roottoleaf way [9].

In this paper, A novel holistic twig join algorithm, called TwigStack+ is proposed in [86]. It is based on holistic twig join guided by extended solution extension to avoid many redundant computations in the call of getNext. It significantly improves the query processing cost, simply because it can check whether other elements can be processed together with current one. The merging phase. Another improvement consists in reducing the cost of queries execution. Indeed there exists very interesting approaches that eliminate the second phase of merging of individual solutions . These algorithms yield no intermediate results. It proposed the first tree pattern solution, called Twig2Stack that avoids any post path join. Twig2Stack is based on a hierarchical stack encoding scheme to compactly represent the twig results [10].

3. PROPOSED SYSTEM ARCHITECTURE

We build a theoretical framework on optimal processing of XML tree pattern queries. We show that "matching cross" is the key reason to result in the sub-optimality of holistic algorithms. Intuitively, matching cross describes a dilemma where holistic algorithms have to decide whether to output useless intermediate result or to miss useful results.

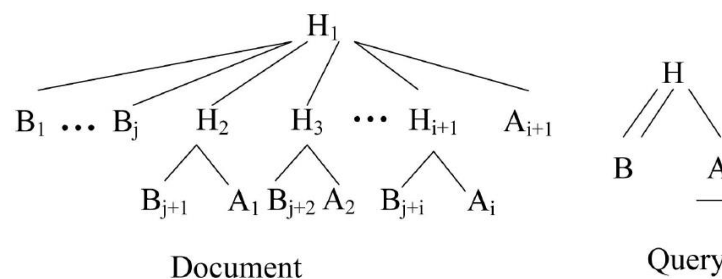

Fig. 3 Proposed scheme of unbounded matching cross.

In order to understand this, let us first consider an XML tree in Fig. 3 and an XPath query "H½:==B_=A" (A is the selected return query node). <B1;Bjþ1;A1;Aiþ1> is a UMC, since I and j maybe greater than the height of XML tree. But, we observe that this UMC still can be efficiently processed by registering the information that H1 has appropriate children B (e.g., B1) and then scanning Bjþ1 and H2. Then, we can get an exact match (H2, Bjþ1, A1) without outputting any possibly useless intermediate path. This example shows that the existence of UMC does not necessarily result in the suboptimality of algorithm. Some UMC still can be solved by buffering limited

information in the main memory. The following definition and lemma show that if there is a mediator node in the UMC, then such UMC can be still processed optimally.

4. HOLISTIC ALGORITHMS

In this section, we propose an algorithm to evaluate an extended XML tree query. The challenge in the algorithm is to achieve a large optimal query class according to aforementioned theorems.

4.1 TreeMatch for Q=;==;_

4.1.1 Data Structures and Notations

There is an input list Tq associated with each query node q, in which all the elements have the same tag name q. Thus, we use eq to refer to these elements. curðTqÞ denotes the current element pointed by the cursor of Tq. The cursor can be advanced to the next element in Tq with the procedure advance ðTqÞ.

4.1.2 Branching Query

There is a set Sq associated with each branching query node q (not each query node). Each element eq in sets consists of a three-tuple ðlabel; bitV ector; outputListÞ. label is the extended Dewey label of eq. bitV ector is used to demonstrate whether the current element has the proper

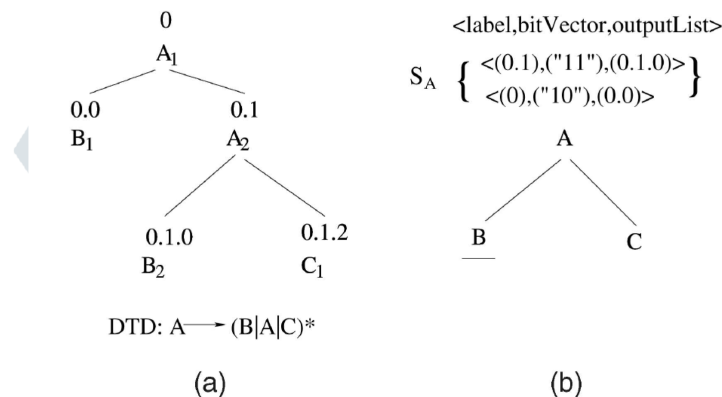

Fig. 4 Illustration to set encoding. (a) an XML tree, and (b) a query with running-time set encoding.

Children or descendant elements in the document. Specifically, the length of bitV ectorðeqÞ equals to the number of child nodes of q. Given a node qc2 childrenðqÞ, we use bitV ectorðeqÞ½qc_ to denote the bit for qc. Specifically, bitV ectorðeqÞ½qc_ ¼ ''1'' if and only if there is an element eqc in the document such that the eq and eqc satisfy the query relationship between q and qc. Finally, output List contains elements that potentially contribute to final query answers. Next, we introduce two properties of elements in output List and bit Vector in details.

4.1.3 TreeMatch

Now we go through Algorithm 1. Line 1 locates the first elements whose pathes match the individual root-leaf path pattern. In each iteration, a leaf node fact is selected by getNext function (line 3). The purpose of lines 4 and 5 is to insert the potential matching elements to outputlist. Line 6 advances the list Tfact and line 7 updates the set encoding. Line 8 locates the next matching element to the individual path. Finally, when all data have been processed, we need to empty all sets in Procedure EmptyAllSets (line 9) to guarantee the completeness of output solutions.

Algorithm 1. Algorithm TreeMatch for class Q=;==;_

1. Locate Match Label(Q);
2. while (:endðrootÞ) do
3. fact ¼ get Next ð top Branching Node Þ;
4. if (fact is a return node)
5. add To Output List ð NAB ð fact Þ; cur ð T fact ÞÞ;
6. advance ð T fact Þ; // read the next element in T fact
7. update Set ð fact Þ; // update set encoding
8. locate Match Label ð Q Þ; // locate next element with
9. matching path
10. empty All Sets ð root Þ;

In Procedure add To Output List ð q; eq i Þ, we add the potential query answer eqi to the set of Seq , where q is the nearest ancestor branching node of qi (i.e., NABðqiÞ ¼ q). Procedure updateSet accomplishes three tasks. First, clean the sets according to the current scanned elements. Second, add e into set and calculate the proper bitVector. Finally, we need recursively update the ancestor set of e. Because of the insertion of e, the bitVector values of ancestors of q need update.

Algorithm getNext (see Algorithm 2) is the core function called in TreeMatch, in which we accomplish two tasks. For the first task to identify the next processed node, Algorithm getNext(n) returns a query leaf node f according to the following recursive criteria: 1) if n is

a leaf node, f ¼ n (line 2); else 2) n is a branching node, then suppose element ei matches node n in the corresponding path solution (if more than one element that matches n, ei is the deepest one by level) (lines 7 and 8), we return fmin such that the current element emin in Tfmin has the minimal label in all ei by lexicographical order (lines 13 and 20).

Algorithm 2. Procedures and Functions in TreeMatch

1. ***Procedure locate Match Label(Q)***
2. ***for each leaf q 2 Q, locate the extended Dewey label eq in list Tq such that eq matches the individual***
3. ***root-leaf path Procedure addToOutputList(q; eqi)***
4. ***for each eq 2 Sq do***
5. ***if (satisfyTreePattern(eqi ,eq))***
6. ***outputListðeqÞ:addðeqi Þ;***
7. ***Function satisfyTreePattern(eqi ,eq)***
8. ***if (bitV ectorðeq; qiÞ ¼ ‘‘1’’) return true;***
9. ***else return false;***
10. ***Procedure updateSet(q; e)***
11. ***cleanSetðq; eÞ;***
12. ***add e to set Sq; //set the proper bitV ector(e)***
13. ***if (:isRootðqÞ ^ (bitV ectorðeÞ ¼ ‘‘1 . . . 1’’))***
14. ***updateAncestorSetðqÞ;***
15. ***Procedure cleanSet(q; e)***
16. ***for each element eq 2 Sq do***
17. ***if (satisfyTreePattern(eq,e))***
18. ***if (q is a return node)***
19. ***addToOutputListðNABðqÞ; eÞ;***
20. ***if (isTopBranchingðqÞÞ***
21. ***if (there is only one element in Sq)***
22. ***output all elements in outputListðeqÞ;***
23. ***else merge all elements in***
24. ***outputListðeqÞ to outputListðeaÞ, where ea ¼ NABðeqÞ;***
25. ***delete eq from set Sq;***
26. ***Procedure updateAncestorSet(q)***
27. ***=_ assume that q0 ¼ NAB(q)_=***
28. ***for each e 2 Sq0 do***
29. ***if (bitV ectorðe; qÞ ¼ 0)***
30. ***bitV ectorðe; qÞ ¼ 1;***
31. ***if (:isRootðqÞ ^ (bitV ectorðeÞ ¼ ‘‘1 . . . 1’’))***
32. ***updateAncestorSetðq0Þ;***
33. ***Procedure emptyAllSets(q)***
34. ***if (q is not a leaf node)***
35. ***for each child c of q do EmptyAllSets(c);***
36. ***for each element e 2 Sq do cleanSet(q; e);***
37. ***Algorithm 3. getNext(n)***
38. ***if (isLeaf(n)) then***
39. ***return n***
40. ***else***
41. ***for each ni 2 NDB(n) do***
42. ***fi ¼ getNextðniÞ***
43. ***if (isBranching(ni) ^:emptyðSni Þ)***
44. ***return fi***
45. ***else ei ¼ maxfpjp 2 MBðni; nÞg***
46. ***end for***
47. ***max ¼ maxargifeig***
48. ***for each ni 2 NDB(n) do***
49. ***if (8e 2 MBðni; nÞ : e 62 ancestors(emax))***
50. ***return fi;***
51. ***endif***
52. ***end for***
53. ***min ¼ minargiffijfi is not a return nodeg***
54. ***for each e 2 MB(nmin; n)***
55. ***if (e 2 ancestorsðemaxÞ) updateSet(Sn; e)***
56. ***end for***
57. ***return fmin***
58. ***end if***

Function MBðn; bÞ

1. ***if (isBranching(n))then***
2. ***Let e be the maximal element in set Sn***
3. ***else***
4. ***Let e ¼ curðTnÞ***
5. ***end if***
6. ***Return a set of element a that is an ancestor of e such***

That a can match node b in the path solution of e to path pattern pn For the second task of getNext, before an element eb is inserted to the set Sb, we ensure that eb is an ancestor (or parent) of each other element ebi to match node b in the corresponding path solutions (line 13). If there are more than one element to match the branching node b, ebi is defined as their deepest (i.e., maximal) element (line 8).

4.1.4 Modules:

The fact that TwigStack is optimal for queries with only A-D relationships can be explained that no matching cross can be found for any XML document with respect to queries with A-D edges. We classify matching cross to bound and unbounded matching cross (BMC and UMC). We develop theorems to show that only part of UMC (i.e. UMC with mediator) can force holistic algorithms to potentially output useless intermediate results. Based on the theoretical analysis, we develop a series of holistic algorithms TreeMatch to achieve a large optimal query class for Q/,//,*. Our main technique is to use a concise encoding to present matching results, which leads to the reduction of useless intermediate results. We conducted an extensive set of experiment on synthetic and real data set for performance comparison. We compared TreeMatch with previous four holistic XML tree pattern matching algorithms. The experimental results show that our algorithm can correctly process extended XML tree patterns, achieving performance speedup for tested queries and data sets, even in their restricted focus. The improvement mainly owes to the reduction of the size of intermediate results.

4.1.5 Optimality of holistic algorithm:

Previous XML tree pattern matching algorithms do not fully exploit the "optimality" of holistic algorithms. TwigStack guarantees that there is no useless intermediate result for queries with only Ancestor-Descendant (A-D) relationships. Therefore, TwigStack is optimal for queries with only A-D edges. Another algorithm TwigStackList enlarges the optimal query class of TwigStack by including Parent-Child(P-C) relationships in non-branching edges. A natural question is whether the optimal query class of TwigStackList can be further improved. Hence, the current open problems include (1) how to identify a larger query class which can be processed optimally and (2) how to efficiently answer a query which cannot be guaranteed to process optimally. This explores the challenges and shows the promise of a novel theoretical framework called "matching cross" to identify a large optimal query class for posing extended XML tree queries.

4.1.4 Return nodes in twig pattern queries:

In a practical application, only part of query nodes belong to return nodes (or called output nodes interchangeably). Take the XPath "//A[B]//C" as an example, only C element and its subtree are answers. The current "modus operandi" is that they first find the query answer with the combinations of all query nodes, and then do an appropriate projection on those return nodes. Such a post-processing approach has an obvious disadvantage: it outputs many matching elements of non-return nodes that are unnecessary for the final results. Here, we develop a new encoding method to record the mapping relationships and avoid outputting non-return nodes.

4.1.6 Modeling of XML data and extended tree pattern query:

An XML database D is usually modeled as a rooted, node labeled tree, element tags and attributes are mapped to nodes in the trees and the edges are used to represent the direct nesting relationships. Our primary focus is on element nodes; and it is not difficult to extend our methods to process the other types of nodes, including attribute and character data. For convenience, we distinguish between query nodes and database nodes by using the term "node" to refer to a query node and the term "element" to refer to a data element in D. An extended tree query Q describes a complex traversal of the XML document and retrieves relevant tree-structured portions of it. The nodes in Q include element tags, attributes and character data. We use "*" to denote the wildcard, which can match any single tree element. There are four kinds of query edges, which are the four combinations between (positive, negative) and (parent-child, ancestor-descendant).

4.1.7 Matching Cross:

"Matching cross" demonstrates the intrinsic reason for the sub-optimality of existing holistic algorithms.The purposes of our study are (i) to provide insight into the characteristics of the holistic algorithms, and thus promotes our understanding about their behaviors; and (ii) to lead to novel algorithms that can guarantee a larger optimal query class and realize better query performance. The existing holistic algorithms consist of two phases: (i) in the first phase, a list of path solutions is output as intermediate path solutions and each solution matches the individual root-to-leaf path expression; and (ii) in the second phase, the path solutions are merged to produce the final answers for the whole twig query. However, for queries with parent-child (P-C) relationships, the state-of-the-art algorithms cannot guarantee that each intermediate solution output in the first phase is useful to merge in the second phase. In other words, many useless intermediate results (i.e. path solutions) may be produced in the first phase, which is called the suboptimality of algorithms.

5. RESULTS AND DISCUSSION

In this section, we present an extensive experimental study of TreeMatch on real-life and synthetic data sets. Our results verify the effectiveness, in terms of accuracy and optimality, of the TreeMatch as holistic twig join algorithms for large XML data sets. These benefits become apparent in a comparison to previously four proposed algorithms TwigStack [3], TJFast [16], OrderedTJ [17], and TwigStack- ListNot [26]. The reason that we choose these algorithms for comparison is that 1) similar to TreeMatch, both TJFast and

TwigStack are two holistic twig pattern matching algorithms. But they cannot process queries with order restriction or negative edges; and 2) OrderedTJ is a holistic twig algorithm which can handle order-based XML tree pattern, but is not appropriate for queries with negative edges; and finally 3) TwigStacklistNot is proposed for queries with negative edges, but it cannot work for ordered queries. Only TreeMatch algorithm can process queries with order restriction, negative edge, and wildcards.

5.1 Experiment Settings and Data Set

We implemented all tested algorithms in JDK 1.4 using the file system as a simple storage engine. We conducted all the experiments on a computer with Intel Pentium IV 1.7 GHz CPU and 2 G of RAM. To offer a comprehensive evaluation of our new algorithms, we conducted experiments on both synthetic and real XML data. The synthetic data set is generated randomly. There are totally seven tags A, B; . . . ; F, G in the data set and tags are assigned uniformly from them. The real data are DBLP (highly regular) and Treebank (highly irregular), which are included to test the two extremes of the spectrum in terms of the structural complexity. The recursive structure in TreeBank is deep (average depth: 7.8 and maximal depth: 36). We can easily find queries on this data set to demonstrate the suboptimality for our tested algorithms.

5.2 Query Class Q=;==;_

In this section, we show the experimental results for query class Q=;==;_. All queries tested in our evaluation are shown in Figs. 5 and 6.

5.2.1 Small Size of Main Memory

In the first experiment, we did not allow the output list in TreeMatch to buffer any elements in the main memory, meaning that any element added to output list should be output to the secondary storage. Then, the requirement for main memory size is quite small. The purpose of this experiment is to simulate the application where the document is extremely large but the available main memory is relatively small. Table 4 shows the number of total output elements (including intermediate and final results) and the corresponding percentage of useful elements. We made the experiments by using three different sizes of random documents. In particular, D1 has 100K nodes and D2 has 500K nodes, and D3 has 1M nodes. From Table 4, we observe that for most of queries, TreeMatch achieves the

TABLE 1 Number of Output Elements (O) and the Percentage (P) of Useful Elements for TreeMatch on Random Data

Query	D1		D2		D3	
	O	P	O	P	O	P
Q1	1321	100%	6576	100%	13290	100%
Q2	3558	100%	177757	100%	35649	100%
Q3	9575	98.8%	95291	99.9%	156954	94.5%
Q4	6635	100%	33055	100%	65691	100%
Q5	296	100%	1313	100%	2782	100%
Q6	7506	100%	94132	100%	127478	99.9%

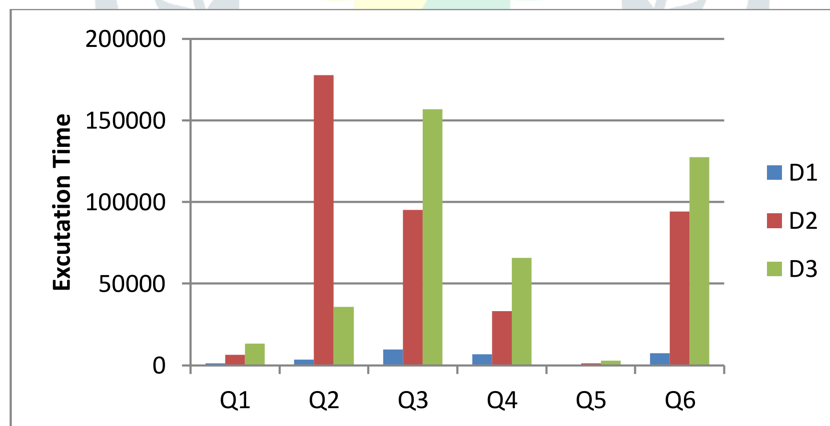

Fig. 5. Execution time of Q=;==;_ on random data. (a) Small memory. (b) Large memory.

5.2.2 Large Size of Main Memory

In the second experiment, we allow the output list to buffer all elements in the main memory. The purpose of this experiment is to simulate the application where the available main memory is large so that a big portion of documents can fit in the main memory. Table 5 shows the maximal number of elements buffered in order to avoid outputting any useless intermediate results. An obvious observation is that Q3 and Q6 need to buffer many elements, but all other queries only need to buffer very small number of elements. This also can be explained thatall queries except Q3 and Q6 belong to the optimal query class. We compared the performance of three algorithms in Figs. 17b and 18a. Obviously, TreeMatch is superior to TwigStack and TJFast, reaching 20 to 95 percent improvement in execution time for all queries.

5.2.3 Medium Size of Main Memory

In most real applications, the main memory size is not so large that the whole document can fit in memory, neither so limited that only the elements in a single path can load in memory. In order to test whether TreeMatch has the ability to fully exploit the available medium size of main memory.

CONCLUSION

In this paper, we presented the survey results of XML based pattern matching algorithms. The TreeMatch algorithms with different query classes are introduced. TreeMatch has an overall good performance in terms of running time and the ability to process extended XML tree patterns (twigs). The We have introduced a notion of matching cross to address the problem of the sub-optimality in holistic XML tree patten matching algorithms. We have identified a large optimal query classes for querie, that is Q/,//,* . Based on these results, we have proposed a new holistic algorithm called TreeMatch to achieve such theoretical optimal query classes. Finally, extensive experiments demonstrate the advantage of our algorithms and verify the correctness of theoretical results.

REFERENCES

[1] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu, and R. Buyya, "Ensuring security and privacy preservation for cloud data services," ACM Computing Surveys, 2016.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50–58, 2010.

[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in Proceedings of the 13th ACM Conference on Computer and Communications Security. ACM, 2006, pp. 79–88.

[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in Advances in CryptologyEurocrypt 2004. Springer, 2004, pp. 506–522.

[5] Z. Ying, H. Li, J. Ma, J. Zhang, and J. Cui, "Adaptively secure ciphertext-policy attribute-based encryption with dynamic policy updating," Sci China InfSci, vol. 59, no. 4, pp. 042 701:1–16, 2016.

[6] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Security and Privacy, 2000. SP 2000. Proceedings. 2000 IEEE Symposium on, 2000, pp. 44–55.

[7] E.-J. Gohet al., "Secure indexes." IACR Cryptology ePrint Archive, vol. 2003, p.216, 2003.

[8] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keywordsearches on remote encrypted data," in Applied Cryptography and Network Security. Springer, 2005, pp. 442–455.

[9] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in Pairing-Based Cryptography– Pairing. Springer, 2007, pp. 2–22.

[10] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in Applied Cryptography and Network Security. Springer, 2004, pp. 31–45.

[11] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in Informationand Communications Security. Springer, 2005, pp. 414–426.
[12] D. Boneh and B. Waters, "Conjunctive, subset, and range qNetwork Security. Springer, 2005, pp. 442

[13] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 361-370, 2001.

[14] J. Lu, T. Chen, and T.W. Ling, "Efficient Processing of XML Twig Patterns with Parent Child Edges: A Look-Ahead Approach," Proc. 13th ACM Int'l Conf. Information and Knowledge Management (CIKM), pp. 533-542, 2004.

[15] J. Lu, T.W. Ling, Z. Bao, and C. Wang, "Extended XML Tree Pattern Matching: Theories and Algorithms," technical report, 2010.

[16] J. Lu, T.W. Ling, C. Chany, and T. Chen, "From Region Encoding to Extended Dewey: On Efficient Processing of XML Twig Pattern Matching," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 193- 204, 2005.

[17] J. Lu, T.W. Ling, T. Yu, C. Li, and W. Ni, "Efficient Processing of Ordered XML Twig Pattern Matching," Proc. 16th Int'l Conf. Database and Expert Systems Applications (DEXA), pp. 300-309, 2005.

[18] M. Moro, Z. Vagena, and V.J. Tsotras, "Tree-Pattern Queries on a Lightweight XML Processor," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 205-216, 2005.

[19] P. ONeil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: Insert-Friendly XML Node Labels," Proc. ACM SIGMOD, pp. 903-908, 2004.

[20] P. Rao and B. Moon, "PRIX: Indexing and Querying XML Using Prufer Sequences," Proc. 20th Int'l Conf. Data Eng. (ICDE), pp. 288-300, 2004.

[21] M. Shalem and Z. Bar-Yossef, "The Space Complexity of Processing XML Twig Queries over Indexed Documents," Proc. 24th Int'l Conf. Data Eng. (ICDE), 2008.

[22] I. Tatarinov, S. Viglas, K.S. Beyer, J. Shanmugasundaram, E.J. Shekita, and C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," Proc. ACM SIGMOD, pp. 204-215, 2002.

[23] H. Wang and X. Meng, "On the Sequencing of Tree Structures for XML Indexing," Proc. 21st Int'l Conf. Data Eng. (ICDE), pp. 372-383, 2005.

[24] H. Wang, S. Park, W. Fan, and P.S. Yu, "ViST: A Dynamic Index Method for Querying XML Data by Tree Structures," Proc. ACM SIGMOD, pp. 110-121, 2003.

[25] W. Wang, H. Wang, H. Lu, H. Jiang, X. Lin, and J. Li, "Efficient Processing of XMLPath Queries Using the Disk-Based F&B Index," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 145-156, 2005.